Full Pipeline Using Spark

Ashwanth Samuel (as7cs)

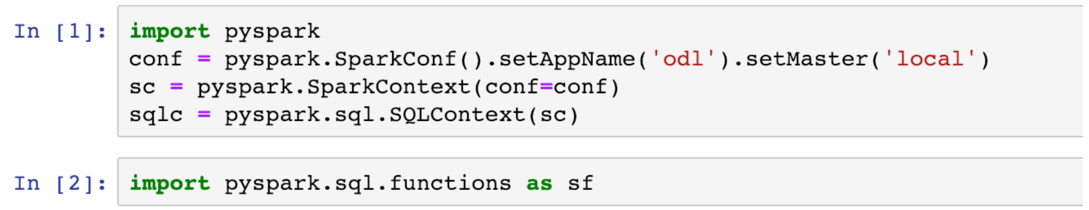
MSDS Class of ‘19

Motivation

- This dataset describes songs from Spotify in 2017 with attributes utilizing the streaming services' API.
- There are 16 attributes including acoustiness, danceability, duration in milliseconds, energy, loudness etc.
- I wanted to run a simple linear regression using Spark and MLlib software and that is detailed here in these slides.

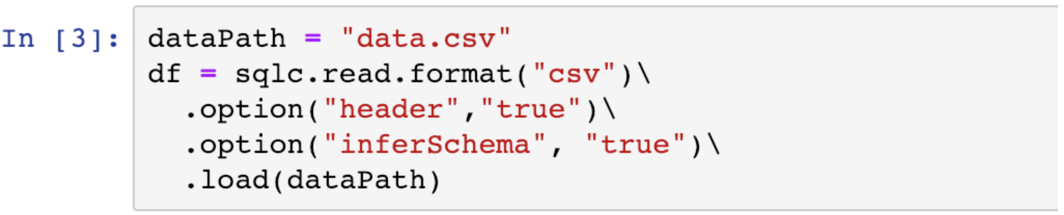
I ultimately decided that I wanted to try and predict the energy of a song from its "speechiness" factor.

Creation of Context

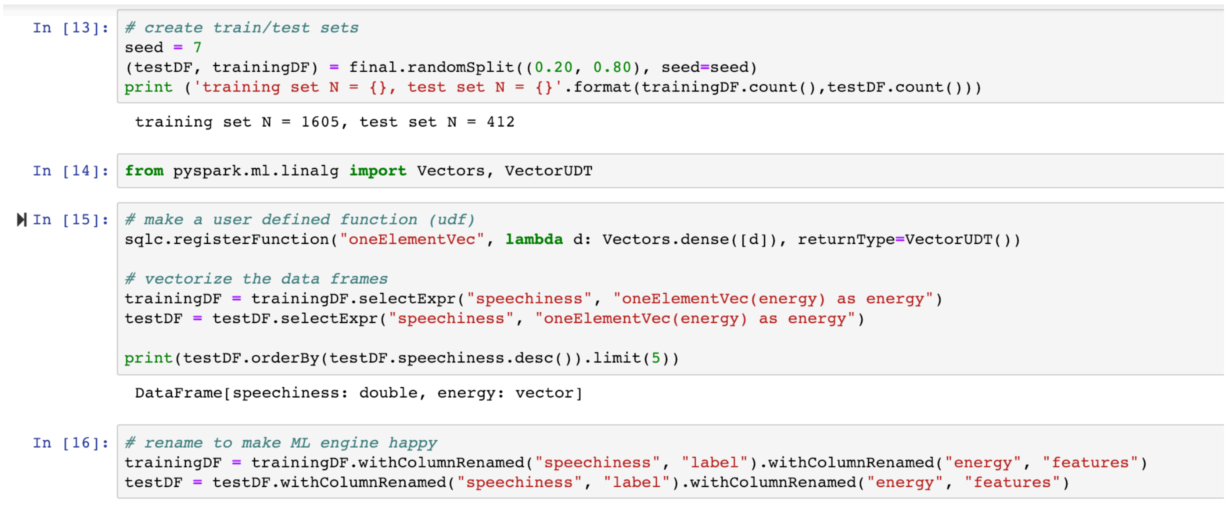


```
In [1]: import pyspark
        conf = pyspark.SparkConf().setAppName('odl').setMaster('local')
        sc = pyspark.SparkContext(conf=conf)
        sqlc = pyspark.sql.SQLContext(sc)

In [2]: import pyspark.sql.functions as sf
```

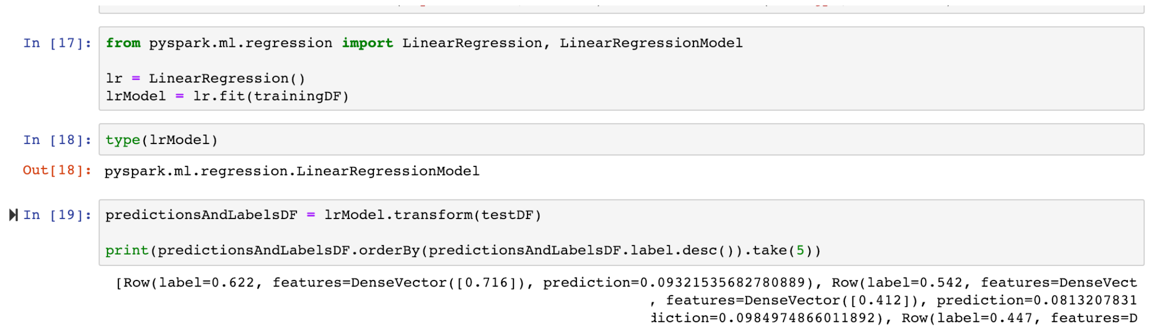
Preparation of Data



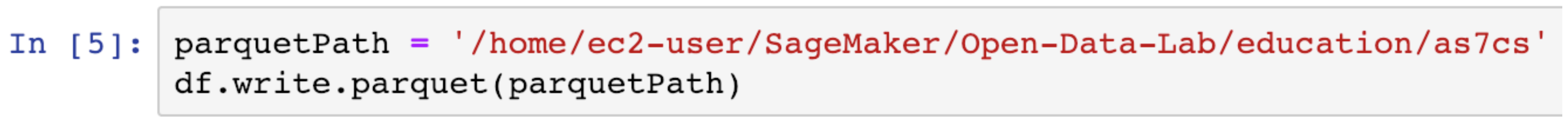
```
In [3]: dataPath = "data.csv"
        df = sqlc.read.format("csv")\
          .option("header","true")\
          .option("inferSchema", "true")\
          .load(dataPath)
```



```
In [13]: # create train/test sets
         seed = 7
         (testDF, trainingDF) = final.randomSplit((0.20, 0.80), seed=seed)
         print ('training set N = {}, test set N = {}'.format(trainingDF.count(),testDF.count()))

          training set N = 1605, test set N = 412

In [14]: from pyspark.ml.linalg import Vectors, VectorUDT

In [15]: # make a user defined function (udf)
         sqlc.registerFunction("oneElementVec", lambda d: Vectors.dense([d]), returnType=VectorUDT())

         # vectorize the data frames
         trainingDF = trainingDF.selectExpr("speechiness", "oneElementVec(energy) as energy")
         testDF = testDF.selectExpr("speechiness", "oneElementVec(energy) as energy")

         print(testDF.orderBy(testDF.speechiness.desc()).limit(5))

          DataFrame[speechiness: double, energy: vector]

In [16]: # rename to make ML engine happy
         trainingDF = trainingDF.withColumnRenamed("speechiness", "label").withColumnRenamed("energy", "features")
         testDF = testDF.withColumnRenamed("speechiness", "label").withColumnRenamed("energy", "features")
```



```
In [17]: from pyspark.ml.regression import LinearRegression, LinearRegressionModel

         lr = LinearRegression()
         lrModel = lr.fit(trainingDF)

In [18]: type(lrModel)

Out[18]: pyspark.ml.regression.LinearRegressionModel

In [19]: predictionsAndLabelsDF = lrModel.transform(testDF)

         print(predictionsAndLabelsDF.orderBy(predictionsAndLabelsDF.label.desc()).take(5))

          [Row(label=0.622, features=DenseVector([0.716]), prediction=0.09321535682780889), Row(label=0.542, features=DenseVect
                                                                            , features=DenseVector([0.412]), prediction=0.0813207831
                                                                            iction=0.0984974866011892), Row(label=0.447, features=D
```

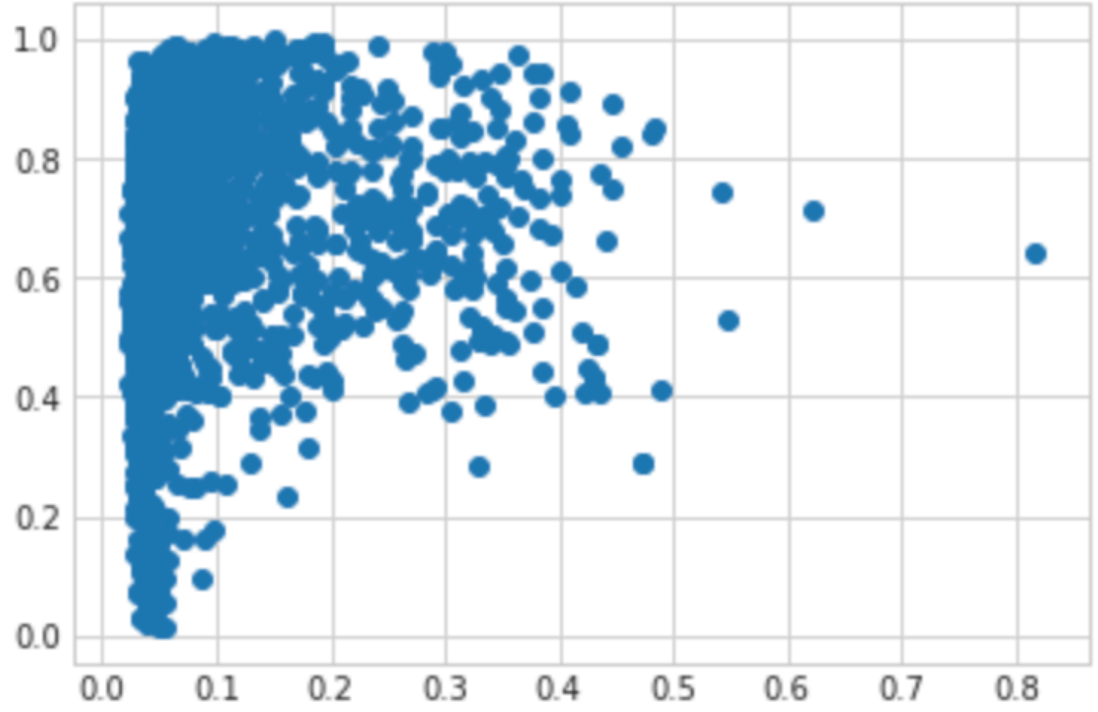


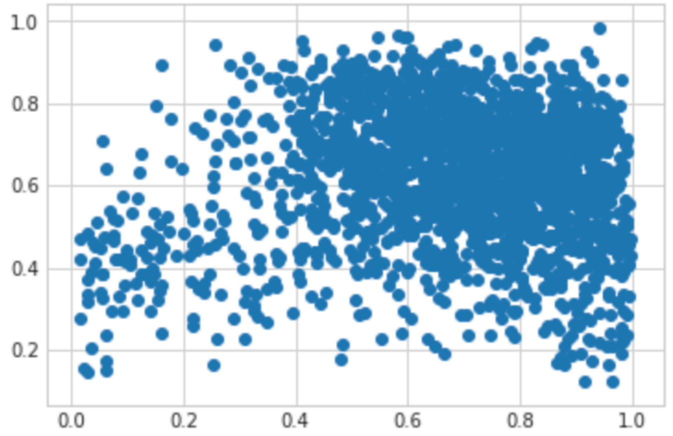
```
In [5]: parquetPath = '/home/ec2-user/SageMaker/Open-Data-Lab/education/as7cs'
        df.write.parquet(parquetPath)
```

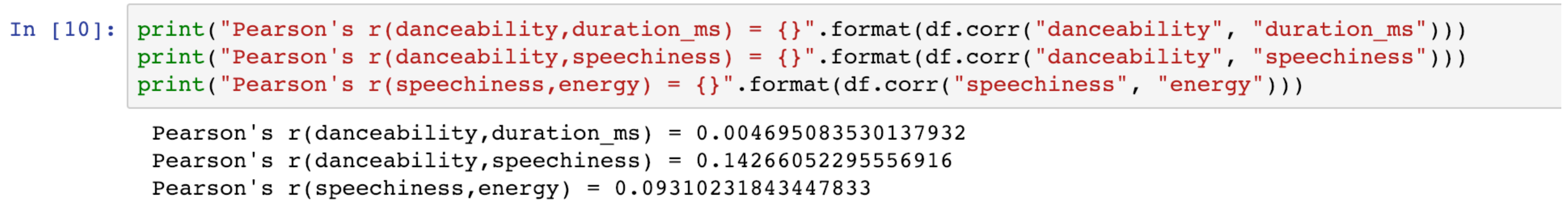
Code

Visuals

The rightgraph is a plot of energy versus danceabilitly. From this we can see that the model is doomed to fail from the start. The bottom visual is a scatter plot of energy and danceability, which shows a little more promise for future analysis.







```
In [10]: print("Pearson's r(danceability,duration_ms) = {}".format(df.corr("danceability", "duration_ms")))
         print("Pearson's r(danceability,speechiness) = {}".format(df.corr("danceability", "speechiness")))
         print("Pearson's r(speechiness,energy) = {}".format(df.corr("speechiness", "energy")))

          Pearson's r(danceability,duration_ms) = 0.004695083530137932
          Pearson's r(danceability,speechiness) = 0.14266052295556916
          Pearson's r(speechiness,energy) = 0.09310231843447833
```